

Voxblox: Building 3D Signed Distance Fields for Planning

Helen Oleynikova, Zachary Taylor, Marius Fehr, Juan Nieto, and Roland Siegwart
Autonomous Systems Lab, ETH Zürich

Abstract—Truncated Signed Distance Fields (TSDFs) have become a popular tool in 3D reconstruction, as they allow building very high-resolution models of the environment in real-time on GPU. However, they have rarely been used for planning on robotic platforms, mostly due to high computational and memory requirements. We propose to reduce these requirements by using large voxel sizes, and extend the standard TSDF representation to be faster and better model the environment at these scales.

We also propose a method to build Euclidean Signed Distance Fields (ESDFs), which are a common representation for planning, incrementally out of our TSDF representation. ESDFs provide Euclidean distance to the nearest obstacle at any point in the map, and also provide collision gradient information for use with optimization-based planners.

We validate the reconstruction accuracy and real-time performance of our combined system on both new and standard datasets from stereo and RGB-D imagery. The complete system will be made available as an open-source library called *voxblox*.

I. INTRODUCTION

Robotic mapping has two main applications: creating high-quality 3D reconstructions for human use and creating maps of an environment that a robot can use for planning. These two types of mapping differ significantly in their requirements: mapping for 3D reconstruction needs high resolutions, small feature sizes, and color, but needs no information about free space. In contrast, in planning applications it is more desirable to use the lowest resolution representation that will still describe the environment relative to the robot size, and distance or gradient information in free space is very valuable.

We present a method to adapt Truncated Signed Distance Fields (TSDFs), a commonly-used map representation for 3D reconstruction, to be able to handle large voxel sizes for planning applications. TSDFs are a sampled implicit surface representation, where space is split into voxels, each of which contains a distance to the nearest surface. They use the projective distance along a sensor ray to estimate the local distance to the surface, which is only a good approximation very close to the surface boundaries [1]. Therefore, the distances are truncated to only a small region around the surface boundaries.

However, existing work on TSDFs has focused on achieving the smallest voxel sizes still possible to process in real-time on a GPU, and the standard methods yield inaccurate and slow reconstructions when applied to large voxel sizes. We explore extensions to the standard formulation to both speed up insertion by an order of magnitude and better preserve the original geometry for large voxel sizes.

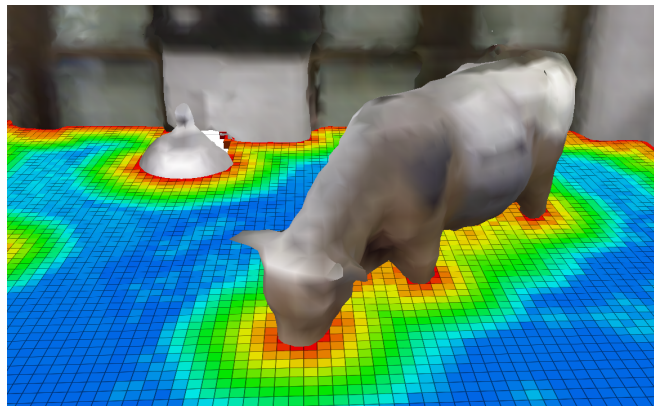


Fig. 1: The output of our proposed system: a mesh model of a life-size fiberglass cow, built from Kinect sensor data, is visualized with a 3D slice of its Euclidean Signed Field. The color represents distance to the surface boundary.

Additionally, for many planning applications, it is helpful or necessary to know the global Euclidean distance to the nearest surface; for example, trajectory optimization methods require collision gradient information inside, near, and outside the object boundaries [2]. A map containing the Euclidean distance to the nearest surface for all points in space is called a Euclidean Signed Distance Field (ESDF) or Euclidean Distance Transform (EDT). We propose a method to build such ESDFs directly from TSDFs, while leveraging the existing distance information near surface boundaries. To the best of the authors' knowledge, we are the first to incrementally build ESDFs from TSDF data, and we use a formulation that allows dynamically changing the map size.

TSDFs are a convenient representation for many applications, especially when mesh output is required. Combining them directly with ESDFs creates a fast and flexible framework for on-board mapping and planning for mobile robots. We also show that we are able to integrate data significantly faster than the commonly used Octomap occupancy map [3]. Even without using an optimization-based planner, we show that using an ESDF for collision checking trajectories leads to far fewer look-ups compared to standard occupancy maps.

Finally, we present a complete system called *voxblox* which combines the improved TSDF with a dynamically-updating ESDF layer, in real-time on a single CPU core. We validate it on real datasets from a variety of sensors, including the EuRoC stereo reconstruction benchmark, KITTI raw stereo datasets, and our own RGB-D indoor dataset. The sample output of the system is shown in Fig. 1.

The contributions of this work are as follows:

- We present a new merging strategy for new data in TSDFs that both speeds up insertion and increases accuracy at large voxel sizes (Section IV)
- We show a method to build an ESDF out of a TSDF and some examples of the advantages of using ESDFs for planning (Section V)
- Make the complete system available open-source (Section VII)
- Compare multiple merging, weighting, and ESDF-building strategies on real-world datasets from stereo and RGB-D sensors (Section VIII)

II. RELATED WORK

This section gives a brief overview of how signed distance fields have been used in both mapping and planning literature and show where our work attempts to bridge the gap between the two.

A. Mapping Literature

Truncated Signed Distance Fields (TSDFs), originally used as an implicit 3D volume representation for graphics, have become a popular tools in 3D reconstruction with KinectFusion [1], which uses the RGB-D data from a Kinect sensor and a GPU adaptation of Curless and Levoy’s work [4], to create a system that could reconstruct small environments in real-time at millimeter resolution.

The main restrictions of this approach is the fixed-size voxel grid, which requires a known map size and a large amount of memory. There have been multiple extensions to to overcome this shortcoming, including using a moving fixed-size TSDF volume and meshing voxels exiting this volume [5], using an octree-based voxel grid [6], and allocating blocks of fixed size on demand in a method called *voxel hashing* [7].

The focus of all of these methods is to output a high-resolution mesh in real-time using marching cubes [8], frequently on GPUs. There has also been work on speeding up these algorithms to run on CPU [6] and even on mobile devices [9]; however, the application is always to create a visually-appealing 3D reconstruction. Instead, our work focuses on creating representations that are accurate and fast enough to use for planning onboard mobile robots, while using large voxels to speed up computations and save memory.

B. Planning Literature

Maps are a crucial part of collision-free planning, and the choice of map representation determines which planners may be used.

Occupancy grids represent the most commonly-used type of map representation for planning in 2D. Elfes *et al.* uses a fixed-size grid, probabilistic model of sensor measurements to model observed and unknown space explicitly [10]. Naively extending occupancy grids to 3D, however, leads to huge memory requirements that quickly become intractable

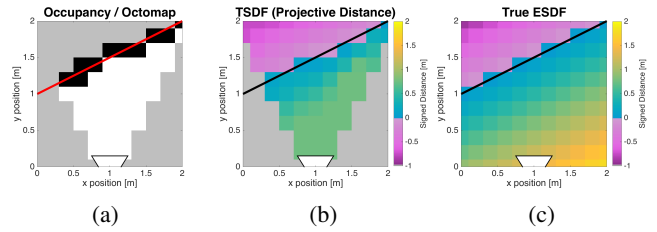


Fig. 2: Comparison of map building strategies from a single sensor scan of a line. (a) shows an occupancy representation, where each cell is either labelled as occupied or free. (b) shows a TSDF, which stores projective (along the sensor ray) distance information close to the object boundary. (c) shows the ground truth ESDF, which represents the true Euclidean distance to the surface at each cell.

for any space larger than a room. Hornung *et al.* proposed using an octree to resolve the scalability issues for 3D maps [3]. Octomap uses a hierarchical data structure to store occupancy probabilities for voxels, using a simplified sensor model.

However, there are planning approaches for which only occupancy information is insufficient. For example, trajectory optimization-based planners, such as CHOMP [11] and TrajOpt [12], require distances to obstacles and collision gradient information. This requires distance information even in free space. This is usually obtained by building a Euclidean Signed Distance Field (ESDF) in batch from another map representation.

Lau *et al.* have presented an efficient method of dynamically building ESDFs out of occupancy maps [13]. Their method exploits the fact that sensors usually observe only a small section of the environment at a time, and significantly outperform batch ESDF building strategies for robotic applications. We extend their approach to be able to build ESDFs out of TSDFs incrementally.

One existing work that combines ESDFs and TSDFs is that of Wagner *et al.*, who use KinectFusion combined with CHOMP for planning for an armed robot [14]. However, instead of updating the ESDF incrementally, they first build a complete TSDF, then convert it to an occupancy grid and compute the ESDF in a single batch operation for a fixed-size volume. In contrast, our incremental approach gives us the ability to maintain an ESDF directly from a TSDF, handle dynamically growing map without knowing the size *a priori*, and is significantly faster than batch methods.

III. SIGNED DISTANCE FIELD DEFINITIONS

This section aims to clarify the notation used in the remainder of the paper and compare multiple ways of building distance fields or occupancy grids from sensor data.

Fig. 2 shows a comparison of how a single sensor scan is represented in occupancy maps, Truncated Signed Distance Fields, and Euclidean Signed Distance Fields.

Fig. 2(a) shows an occupancy map, which are commonly used for planning. Each voxel represents an area of 3D space,

and has one value associated with it – the probability of occupancy.

Fig. 2(b) shows a Truncated Signed Distance Field (TSDF) which is an implicit surface representation, where the zero-crossings represent locations of surfaces, positive values indicate free space, and negative values indicate occupied space behind a surface. The distances in a TSDF are taken along each sensor ray, and truncated to a small radius around the surface boundary. Each voxel in a TSDF also has an associated weight, which allows more accurate merging of multiple scans into a single field [4].

The map shown in Fig. 2(c) is a Euclidean Signed Distance Field (ESDF) which contains the true Euclidean to the nearest surface. This representation is needed or useful in many planning applications, as collision checks are significantly sped up by knowing global distances, and collision cost gradients are available everywhere in the map, not only on object boundaries [11].

The key difference between the ESDF and the TSDF is how the distances are calculated – TSDFs used *projective* distances, which significantly speed up computation time, and make a good approximation to Euclidean distances very near to the surface boundaries [1]. This approximation is part of the reason TSDFs are usually only computed up to a small *truncation distance* from the isosurface.

A more extensive comparison of the three representations and quantitative results are available in [15].

IV. INTEGRATING SCANS INTO TSDFs

Choices in how to build a TSDF out of sensor data can have a large impact on both integration speed and the accuracy of the resulting reconstruction. Here we present three parts of our approach: the weighting, which improves accuracy over the standard model most literature uses while being general enough to describe most vision-based sensors; the merging, which yields significant speed-ups for integrating at large voxel sizes; and an anti-grazing heuristic, which leads to accuracy improvements in certain circumstances.

A. Weighting

The standard strategy to integrate a new scan into a TSDF is to ray-cast from the sensor origin to every point in the sensor data, and update the distance and weight estimates along this ray. The choice of weighting function can have a strong impact on the accuracy of the resulting reconstruction, especially for large voxels, where thousands of points may be merged into the same voxel *per scan*.

KinectFusion discussed using weights based on θ , the angle between the ray from the sensor origin and the normal of the surface, however then stated that a constant weight of 1 was sufficient to get good results [1]. Bylow *et al.* later explored how this weight should behave behind the isosurface, and what function it should drop off with behind the surface boundary [16]. However, their work still used a constant weight – that is, all rays are weighed equally up to the truncation distance, and this is the most common approach in other literature [1], [5], [7], [17].

The general equations governing the merging are based on the existing distance and weight values of a voxel, D and W , and the new update values from a specific point observation in the sensor, d and w , where d is the distance from the *surface boundary*. Given that \mathbf{x} is the position of the current voxel, \mathbf{p} is the position of a 3D point in the incoming sensor data, \mathbf{s} is the sensor origin, and $\mathbf{x}, \mathbf{p}, \mathbf{s} \in \mathbb{R}^3$, the updated D distance and W weight values of a voxel at \mathbf{x} will be:

$$d(\mathbf{x}, \mathbf{p}, \mathbf{s}) = \|\mathbf{p} - \mathbf{x}\| \text{sign}((\mathbf{p} - \mathbf{x}) \bullet (\mathbf{p} - \mathbf{s})) \quad (1)$$

$$w_{\text{const}}(\mathbf{x}, \mathbf{p}) = 1 \quad (2)$$

$$D_{i+1}(\mathbf{x}) = \frac{W_i(\mathbf{x})D_i(\mathbf{x}) + w(\mathbf{x}, \mathbf{p})d(\mathbf{x}, \mathbf{p})}{W_i(\mathbf{x}) + w(\mathbf{x}, \mathbf{p})} \quad (3)$$

$$W_{i+1}(\mathbf{x}) = \min(W_i(\mathbf{x}) + w(\mathbf{x}, \mathbf{p}), W_{\max}) \quad (4)$$

There has also been work on building empirical sensor models for RGB-D sensors, which found that the σ of a single ray measurement varied predominantly with z^2 [18], where z is the depth of the measurement in the camera frame ($z = \|\mathbf{p} - \mathbf{s}\|$). It has been shown that the sensor σ can be used in the weighting equations by substituting w with $\frac{1}{\sigma^2}$ to yield (3), (4) [19]. Based on these findings, we have chosen a simplified weighting model which represents the physics of most vision-based sensors better than standard constant weights (2):

$$w(\mathbf{x}, \mathbf{p}) = \frac{1}{z^2} \quad (5)$$

This allows us to have weights that are more physically meaningful than $w = 1$, while still largely fitting the sensor model of both stereo and projected-light based sensors.

B. Merging

One key drawback of TSDFs for mobile robot applications is the large computational resources required. However, as planning applications do not require a high level of detail in the maps, since they will be largely used for collision checks, we can significantly reduce computational requirements by using large voxel sizes. To the best of our knowledge, no work has addressed the problem of building TSDFs with large voxels, and most existing work focuses on voxel sizes in the millimeter range [1], [7], while we focus sizes on the scale of tens of centimeters.

The key consideration with large voxel sizes is the number of points from a scan that project to the same voxel: while with millimeter-scale voxels, this may be on the order of ones or tens, for 20 cm voxels and a high-resolution RGBD camera, the number may be in the thousands. We exploit this for a significant speedup by designing a strategy that only performs raycasts once per voxel.

For each point in the original scan, its location is projected into the voxel grid, and stored in a multi-map indexed by voxel position. After all points have been grouped by voxel, we merge all points terminating in the same voxel together, using the same merging strategy as in (3) and (4). Ray-casting is then performed only once for each combined sensor measurement, and all voxels in the ray are updated

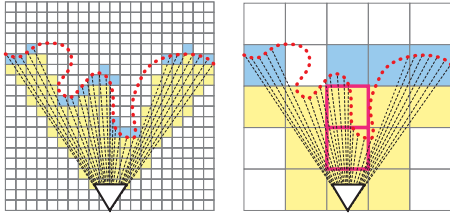


Fig. 3: A single scan of an object (red) at two different voxel resolutions. On the left, a small number of points in the original scan map to each voxel. On the right, large numbers of points and rays map to a single voxel, causing *grazing*, especially in the voxels highlighted in pink: there are both rays with a high distance and those terminating in the voxel. As the voxel distance is a weighted average of all measurements, this will distort the surface geometry.

using the averaged distance and accumulated weight of all measurements within.

Since all measurements are still taken into account, and their weights and distances are combined as usual, this leads to a similar reconstruction result while being up to orders of magnitude faster than the naive raycasting approach.

C. Anti-Grazing

The disadvantage of having many points in the same voxel is an effect we refer to as *grazing*: where a voxel containing a surface boundary may also have rays passing through it toward far away surfaces. An example of this effect is shown in Fig. 3: on the right, the voxels highlighted in pink, half of the rays terminate within the voxel and the other half have a large distance, distorting the surface geometry, as the final distance estimate is a weighted average of all measurements. This effect is especially prominent when the scene contains thin structure or fine features relative to the voxel size.

Our proposed solution is to extend the merging algorithm presented above with a check against grazing.

When updating free space in front of a surface, we first check whether the voxel already belongs to a surface boundary by searching for its index in the multi-map described above. If the voxel already contains a surface, then the free-space measurement is discarded.

This allows us to preserve all surface measurements while discarding free space measurements that would potentially distort the surface estimate.

V. ESDF FROM TSDF

In this section we discuss how to build a Euclidean Signed Distance Field (ESDF) for planning out of a TSDF built from sensor data.

We base our approach on the work of Lau *et al.*, who present a fast algorithm for dynamically updating ESDFs from occupancy maps [13]. We extend their method to work on TSDFs as input data, and additionally allow the ESDF map to dynamically change size.

Building an ESDF can be done in batch, where all voxels are added to the map at once, and then voxels with the lowest

distance to obstacles propagate their distances outwards throughout the map. This approach requires only one queue: *lower*, which keeps indices of the voxels that have been updated. When a voxel exits the *lower* queue, it checks whether any of its neighbors would have their distances lowered through the voxel, and if so, updates their values and inserts them into the queue.

Building an ESDF dynamically allows us to exploit the fact that sensors tend to only see small parts of the environment at any given time, and therefore only a small part of the ESDF needs to be updated. However, this requires additional book-keeping, as voxels that were occupied may become free, and since the *lower* queue only allows decreasing distances, the distance values would become incorrect. This necessitates adding a *raise* queue, which clears voxels back to the maximum value of the ESDF, and adds any neighbors that had the current voxel as a parent back into the queue. A parent relationship is established any time a voxel lowers another voxel's distance value.

Our approach requires several differences from [13]: our voxels do not have occupied or unoccupied states, but instead estimates of the distance function near the surface, and our map is not a fixed size but instead may grow dynamically.

The first change is to introduce a fixed band f in the ESDF: these are voxels that are within the truncation distance δ in the TSDF, and therefore contain good estimates of the distance to the nearest surface already [15]. These voxels take their value from the TSDF and may not be modified by the *raise* or *lower* queue.

As our map may grow dynamically, global voxel indices may no longer map to normal integer values; therefore, we store only the direction toward the parent rather than the full global voxel index of the parent.

Finally, since new voxels may enter the map at any time, we use an observed flag o to keep track of which voxels in the ESDF have already been inserted in the map. A voxel is considered observed if its weight in the TSDF is above a small threshold. We then use this in line 17 of Algorithm 1 to do a crucial part of book-keeping for new voxels: adding all of their neighbors into the *lower* queue, so that the new voxel will be updated to a valid value.

These differences are shown in Algorithm 1, which shows the function of propagating updated values from the TSDF map M_t to the ESDF map M_e . PROCESSRAISEQUEUE and PROCESSLOWERQUEUE remain largely the same as in [13], except that voxels in the fixed band do not get modified.

We also use quasi-Euclidean distance, rather than full Euclidean distance, to reduce computation time, which bounds the worst-case error *at large distances from obstacles* to 8% [20].

Our approach incorporates a bucketed priority queue to keep track of which voxels need updates, with a priority of $|d|$. In the results, we compare three different variants: a FIFO non-priority queue, a priority queue with single insert (where the voxel can only be in the queue once, with priority from its first insert), and a priority queue with multiple inserts [13] (where voxel may enter the queue many times,

Algorithm 1 Updating ESDF from TSDF

```

1: function PROPAGATE( $M_t, M_e$ )
2:   for  $\mathbf{x} \in M_{t,u}$  do ▷ Updated subset of  $M_t$ 
3:     if  $|M_t(\mathbf{x})| \leq \delta$  then
4:       if  $M_e(\mathbf{x}) > M_t(\mathbf{x})$  or not  $o(\mathbf{x})$  then
5:          $M_e(\mathbf{x}) \leftarrow M_t(\mathbf{x})$ 
6:         INSERT(lower,  $\mathbf{x}$ )
7:       else
8:          $M_e(\mathbf{x}) \leftarrow M_t(\mathbf{x})$ 
9:         INSERT(raise,  $\mathbf{x}$ )
10:    else
11:      if  $|M_e(\mathbf{x})| < \delta$  and  $o(\mathbf{x})$  then
12:         $M_e(\mathbf{x}) \leftarrow \text{sign}(M_t(\mathbf{x}))d_{\max}$ 
13:        INSERT(raise,  $\mathbf{x}$ )
14:      else if not  $o(\mathbf{x})$  then
15:         $M_e(\mathbf{x}) \leftarrow \text{sign}(M_t(\mathbf{x}))d_{\max}$ 
16:        INSERT(lower,  $\mathbf{x}$ )
17:        INSERTNEIGHBORS(lower,  $\mathbf{x}$ )
18:  PROCESSRAISEQUEUE(raise)
19:  PROCESSLOWERQUEUE(lower)

```

and so its lowest priority is used).

VI. COMPUTATION ADVANTAGES OF ESDFS FOR PLANNING

ESDFs contain global information about obstacles in a map, which makes them useful for planning. One of the key advantages is the availability of gradient information cheaply (one trilinear interpolation), even inside objects or far from object boundaries. This is necessary for many planning algorithms, especially trajectory optimization-based methods, which require gradient information about the collision costs at every point in a trajectory [2].

However, even without exploiting the special structure of ESDFs in the planner, all planning methods can benefit from significant speed-ups in collision checking speed, under some mild assumptions.

Spheres are a common choice for approximating simple robots, as they are rotation invariant in all axes. It is also common practice in manipulation literature to model complex armed robots as sets of overlapping spheres [2], [14].

Therefore, we think it is a reasonable task to compare the number of lookups necessary to collision-check a spherical robot through an arbitrary trajectory in 3D space. In the case of performing this look-up in a standard occupancy map, we must check every voxel that intersects the sphere. A single collision check of a robot located at \mathbf{x} will therefore require

$$n_o(\mathbf{x}) = \left\lceil \frac{\frac{4}{3}\pi r^3}{v} \right\rceil \quad (6)$$

map lookups, where r is the radius of the robot sphere and v is the voxel size (dimension on a single side).

If $\mathbf{x} = f(t)$ on $t \in [0, t_{\max}]$ defines a trajectory, then we need to check multiple positions along the path to guarantee that the complete trajectory is collision-free. Some naive choices for sampling t may be sampling with a constant Δt , or with a constant $\Delta \|\mathbf{x}\|$ along the arc length of the path.

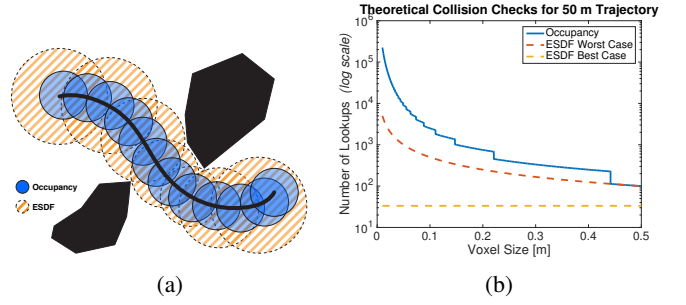


Fig. 4: A comparison of the number of sphere look-ups necessary to collision-check a trajectory: (a) in an occupancy map (blue) versus ESDF (orange stripes), while (b) shows the theoretical bounds on number of lookups. Note that the ESDF checks are bounded between a lower bound (assuming all free space, up to maximum computation distance) and an upper bound where the space is occupied until the robot boundaries. Discontinuities in the occupancy plot are from the ceiling operator in (7).

However, since collision checking is generally expensive, it would make the most sense to minimize the number of poses that need to be checked by checking only overlapping spheres, as shown in Fig. 4(a). This way, it is possible to check an entire trajectory $f(t)$ with total arc length l in only:

$$n_o(f(t)) = \left\lceil \frac{\frac{4}{3}\pi r^3 - \frac{1}{2}\frac{5}{12}\pi r^3}{v} \right\rceil \frac{l}{r} = \left\lceil \frac{\frac{9}{8}\pi r^3}{v} \right\rceil \frac{l}{r} \quad (7)$$

operations, where the subtracted term is half of the volume of the sphere overlap.

In contrast, an ESDF allows collision-checking an entire sphere in only one lookup at its center. If the distance at the center is $d(\mathbf{x}) \geq r$, then the sphere is free; otherwise it is occupied. This also means that the worst case scenario for collision-checking an entire trajectory in an ESDF is

$$\max n_e(f(t)) = \frac{l}{r}. \quad (8)$$

However, the best-case scenario happens if $d(\mathbf{x}) \gg r$ and $d(\mathbf{x}) = d_{\max}$, where d_{\max} is the maximum computed distance of the ESDF:

$$\min n_e(f(t)) = \frac{l}{d_{\max}}. \quad (9)$$

This theoretical comparison is shown in Fig. 4(b), where it is evident that an ESDF will always require fewer operations than an occupancy map for collision checking (note the log scale in the y axis).

VII. SYSTEM IMPLEMENTATION

The C++ mapping library implementing these algorithms, called **voxblox**, is available open-source. The library focuses on flexibility and extensibility for prototyping: it is simple to add new voxel types, *layers* containing these voxels, and *integrators* that merge new data (from sensors or other layers) into these voxel layers. A system diagram of the

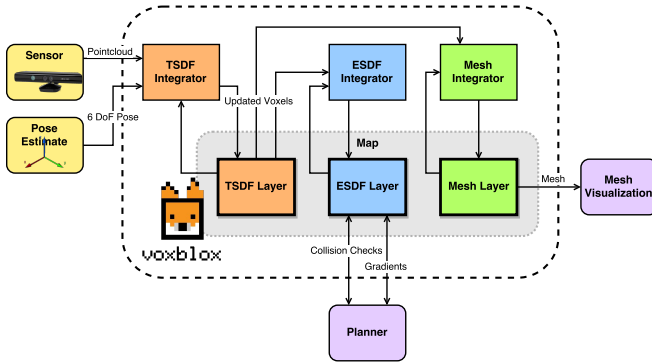


Fig. 5: System diagram for voxblox, showing how the multiple map layers (TSDF, ESDF, and mesh) interact with each other and with incoming sensor data through integrators.

configuration used for these experiments, including all the layers and integrators used, is shown in Fig. 5.

As the underlying data structure, we follow the voxel hashing approach of Niessner *et al.* [7]. Each layer contains a hash map of *blocks*, which are simply fixed-size arrays of voxels. The blocks are allocated on-demand when new data enters the map, and allow dynamic growth of the map while minimizing memory usage.

The implementation of the integrators is single-threaded and entirely on CPU (for experiments, quad-core Intel i7 at 2.5 GHz) for ease of prototyping, though it can be easily parallelized. While the library itself is stand-alone with minimal requirements, we also provide ROS bindings for easy import of data and visualization.

VIII. EXPERIMENTAL RESULTS

In this section we validate the algorithms presented above on three real-world datasets, taken with both projected light and stereo sensors, at a variety of scales. All datasets are publically available. Results on all datasets are shown in the video attachment.

A. Datasets

1) *Cow*: The cow dataset features several objects including a large fiberglass cow and a mannequin (not shown) in a small room. It is taken with the original Microsoft Kinect, uses pose data from a Vicon motion capture system, and the ground truth is from a Leica TPS MS50 laser scanner with 3 scans merged together. It is made publically available with this publication.

2) *EuRoC*: The EuRoC dataset is a public benchmark on 3D reconstruction accuracy [21], in a medium-sized room filled with objects. It is taken with a narrow-baseline stereo grayscale stereo sensor, using Vicon fused with IMU as pose information, and also Leica TPS MS50 scans as structure ground truth. We use the V1_01_easy dataset for experiments.

3) *KITTI*: KITTI is a well-known robotics benchmark focused on autonomous driving applications [22]. We use the raw datasets, using the color stereo pair for reconstruction, fused IMU and GPS as pose, and the stitched

together LIDAR scans as structure ground truth. We use the 2011_09_26_drive_0035 dataset.

B. Surface Reconstruction Accuracy

In order to verify that our merging strategy scales well with larger voxel sizes, and that the merging without grazing strategy shows improvements on fine structure, we validate our TSDF reconstructions against the structure ground truth for our datasets. While the focus of our work is not to create the best or most accurate surface reconstructions, planning applications require that the underlying geometry is well presented by our low-resolution models. Distortions in the overall object geometry lead to inaccuracies in the distance estimates in the resultant ESDF, and may therefore lead to infeasible plans.

We evaluate the accuracy of our reconstruction by projecting each point in the ground truth pointcloud into the TSDF, performing trilinear interpolation to get the best estimate of the distance at that point, and taking that distance as an error. If the reconstruction was perfect, the ground truth points would land perfectly on the zero iso-surface. In cases where the distance is outside the truncation distance, we take the truncation distance as the error, and do not consider unknown voxels in the calculation.

Qualitative comparison are shown on the cow dataset in Fig. 6, compared to the ground truth cow silhouette. As can be seen, constant weighting significantly distorts the geometry of the cow at larger voxel sizes: the head is no longer in the correct position, and the rear legs are gone entirely, which will lead to incorrect distance estimates in the ESDF. Our weighting strategy improves the accuracy of the overall shape of the cow, and the anti-grazing filter further preserves larger features on the cow.

Quantitative comparison over multiple datasets and multiple voxel sizes can be seen in Table I. We chose the voxel sizes to use for the comparisons based on the map size and range of the sensor. As can be seen, both the weighting strategy with $w = 1/z^2$ and weighting with anti-grazing filter outperform constant ($w = 1$) weighting found in most literature. Anti-grazing helps substantially with the cow dataset, but makes little or negative difference in EuRoC dataset. This is due to the structure of scene in EuRoC: there are almost no thin/small features and most objects are large, flat, and up against big walls. KITTI, on the other hand, also sees improvements from the anti-grazing strategy, as there are many fine details relative to the voxel size, like trees and mailboxes.

Finally, a comparison of the timings between various merging strategies and against Octomap is shown in Fig. 7. While Octomap with the same merging strategy as discussed in Section IV-B is already significantly faster than normal raycasting Octomap, it is still substantially slower than our TSDF approach. This is due to the hierarchical data structure: as the number of nodes in the Octomap grows larger, lookups in the tree get slower; with voxel hashing [7], the lookups remain $\mathcal{O}(1)$. Merging leads to significant speeds up, especially with larger voxel sizes (as more points project into the

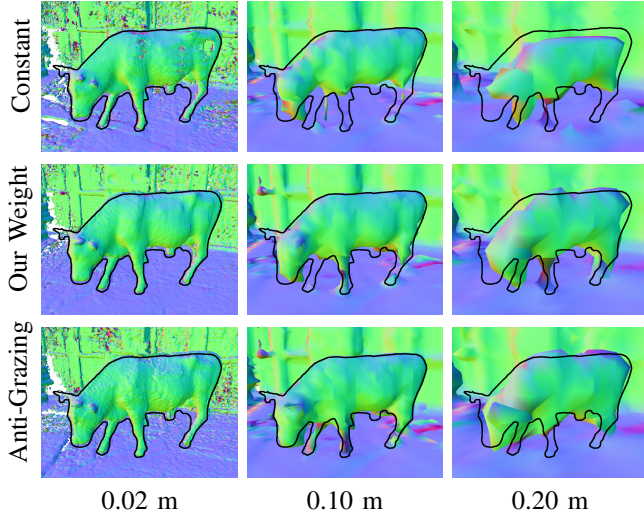


Fig. 6: Qualitative comparisons of weighting/merging strategies on the cow dataset, colored by normals and with the object outline from ground truth overlaid. As can be seen, especially at large voxel sizes, our weighting strategy distorts the structure less (lines up better with ground truth outline), especially with the anti-grazing filter.

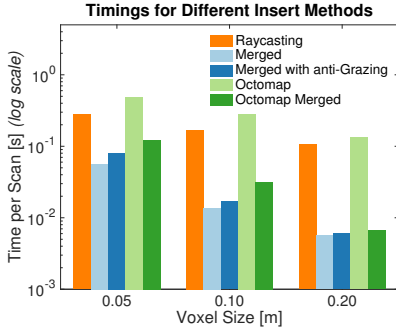


Fig. 7: Timing results for different merging strategies on the EuRoC dataset. Our approach is up to 20 times faster than standard raycasting into a TSDF, and up to 2 times faster than even merged Octomap insertions. Note log scale.

same voxel). Anti-grazing also has a slight negative impact on performance (especially at tiny voxel size, as it adds an additional search operation), but is otherwise comparable. Overall, we show that using our merging strategy makes using TSDFs feasible on a single CPU core, making it suitable for real-time mapping and planning applications.

C. ESDF Construction

To evaluate our incremental ESDF construction strategy, we compare the average time to incorporate new data into the ESDF using a batch approach and our incremental approach, with multiple different queueing methods, as discussed in Section V. It can be seen in Fig. 8 that the building the ESDF incrementally leads to an order of magnitude speedups over the entire dataset, and that at large voxel sizes, using a single-insert priority queue is also significantly faster than

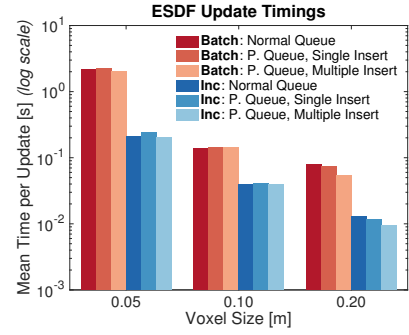


Fig. 8: Timing results for updating ESDF in batch and incrementally, with different queueing strategies on the EuRoC dataset. The normal non-priority queue performs best for small voxel sizes, and at large voxel sizes, there is a significant speed-up from using a single-insert priority queue. Note the logarithmic time scale.

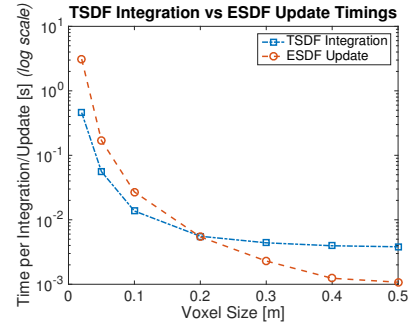


Fig. 9: Timings results for integrating new data into the TSDF compared to propagating new TSDF updates to the ESDF on the EuRoC dataset. At small voxel sizes, TSDF integration is faster, but flattens out at large voxel sizes as the amount of sensor data does not decrease, while ESDF timings continue to decrease.

using a normal FIFO queue.

We also compare the integration time of the TSDF with update time of the ESDF layer in Fig. 9. Though for small voxel sizes, the ESDF update is slower than integrating new TSDF scans, at large enough voxels (here, $v = 0.20$ m), the TSDF integration time flattens out while the ESDF update time keeps decreasing. Since the number of points that need to be integrated into the TSDF does not vary with the voxel size, projecting the points into the voxel map dominates the timings for large voxels.

Therefore, our system is fast enough to build both TSDFs and ESDFs in real-time on a single CPU, enabling its use for real-time onboard planning.

IX. CONCLUSIONS

In this paper, we proposed that Truncated Signed Distance Fields (TSDFs) can be a good environment representation for planning applications, especially when combined with Euclidean Signed Distance Fields (ESDFs). We proposed that for this application, the maps need to use larger voxel sizes than they have in existing literature, showed that the standard

Dataset	Voxel Size [m]	Constant Weight		Our Weight		Our Weight with Anti-Graze		Unknown [frac.]	Memory [MB]
		RMS Error [m]	Insert Time [sec]	RMS Error [m]	Insert Time [sec]	RMS Error [m]	Insert Time [sec]		
EuRoC	0.05	0.0753	0.0554	0.0745	0.0567	0.0781	0.0800	0.676	28.104
	0.10	0.1468	0.0155	0.1462	0.0138	0.1526	0.0170	0.637	6.546
	0.20	0.2676	0.0073	0.2600	0.0056	0.2642	0.0060	0.605	1.575
KITTI	0.20	0.3020	0.5280	0.2776	0.5078	0.2766	0.9346	0.847	30.614
	0.25	0.3651	0.3552	0.3303	0.3477	0.3294	0.5299	0.811	20.179
	0.50	0.6701	0.1176	0.5906	0.0981	0.5795	0.1263	0.652	4.331
Cow	0.02	0.0329	1.3158	0.0325	1.2020	0.0329	2.7645	0.215	512.507
	0.10	0.1402	0.0549	0.1357	0.0539	0.1376	0.0664	0.076	14.076
	0.20	0.2430	0.0310	0.2316	0.0323	0.2261	0.0326	0.056	3.741

TABLE I: Structure reconstruction results on all datasets with multiple voxel sizes. The unknown fraction is the fraction of the ground truth points that mapped to unknown voxels, and the memory is the total memory required to store the TSDF.

formulation of TSDFs is slow and inaccurate under these conditions, and suggested a different weighting and merged raycasting method to overcome these flaws.

We also extended existing incremental ESDF-building methods to work on TSDF input, while leveraging the existing distance information, and proposed adaptations that allow the map to grow dynamically in size. This allows our combined mapping system to be able to scale to large environments, and remove the requirement to know the complete map size *a priori*.

Finally, we validated our complete system, called **voxblox**, on a number of publically-available datasets. We showed that the structure reconstruction accuracy of our approach is better than the standard formulation against structure ground truth, and that our method is also up to 20 times faster. We also presented timing data for ESDF construction, showing that our incremental method is significantly faster than doing the operations in batch, and compared multiple data structures to use for the update queue.

In conclusion, we demonstrated that building both the TSDF and ESDF can be done in real-time on a single CPU core from dense vision-based data, enabling its use for online planning applications.

REFERENCES

- [1] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *IEEE international symposium on Mixed and augmented reality (ISMAR)*, IEEE, 2011.
- [2] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2009.
- [3] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, 2013.
- [4] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of SIGGRAPH*, ACM, 1996.
- [5] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, "Kintinuous: Spatially extended kinectfusion," in *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.
- [6] F. Steinbrucker, J. Sturm, and D. Cremers, "Volumetric 3d mapping in real-time on a cpu," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014.
- [7] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3d reconstruction at scale using voxel hashing," *ACM Transactions on Graphics (TOG)*, 2013.
- [8] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *SIGGRAPH*, ACM, 1987.
- [9] M. Klingensmith, I. Dryanovski, S. Srinivasa, and J. Xiao, "Chisel: Real time large scale 3d reconstruction onboard a mobile device," in *Robotics Science and Systems (RSS)*, 2015.
- [10] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, 1989.
- [11] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "Chomp: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, 2013.
- [12] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, 2014.
- [13] B. Lau, C. Sprunk, and W. Burgard, "Improved updating of euclidean distance maps and voronoi diagrams," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2010.
- [14] R. Wagner, U. Frese, and B. Baumli, "3d modeling, distance and gradient computation for motion planning: A direct gpgpu approach," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2013.
- [15] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart, "Signed distance fields: A natural representation for both mapping and planning," in *RSS Workshop on Geometry and Beyond*, 2016.
- [16] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers, "Real-time camera tracking and 3d reconstruction using signed distance functions," in *Robotics: Science and Systems (RSS)*, vol. 9, Robotics: Science and Systems, 2013.
- [17] O. Kahler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. Torr, and D. Murray, "Very high frame rate volumetric integration of depth images on mobile devices," *IEEE Transactions on Visualization and Computer Graphics*, 2015.
- [18] C. V. Nguyen, S. Izadi, and D. Lovell, "Modeling kinect sensor noise for improved 3d reconstruction and tracking," in *International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*, IEEE, 2012.
- [19] O. Kahler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. Torr, and D. Murray, "Probabilistic multi-sensor fusion based on signed distance function," in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2016.
- [20] U. Montanari, "A method for obtaining skeletons using a quasi-euclidean distance," *Journal of the ACM (JACM)*, 1968.
- [21] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research (IJRR)*, 2016.
- [22] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.